Linear Programming by Solving Systems of Differential Equations Using Game Theory

Daniel Ciuiu

Technical University of Civil Engineering, Bucharest, Romania,
Romanian Institute for Economic Forecasting

June 2009

Linear Programming by Solving Systems of Differential Equations Using Game Theory

Daniel Ciuiu [1,2]

[1] Department of Mathematics and Computer Science,
Technical University of Civil Engineering, Bucharest, Romania.

[2] Romanian Institute for Economic Forecasting, Bucharest, Romania.
e-mail: dciuiu@yahoo.com

***Abstract*— In this paper we will solve some linear programming problems by solving systems of differential equations using game theory.**

The linear programming problem must be a classical constraints problem or a classical menu problem, i.e. a maximization/minimization problem in the canonical form with all the coefficients (from objective function, constraints matrix and right sides) positive.

Firstly we will transform the linear programming problem such that the new problem and its dual have to be solved in order to find the Nash equilibrium of a matriceal game. Next we find the Nash equilibrium by solving a system of differential equations as we know from evolutionary game theory, and we express the solution of the obtained linear programming problem (by the above transformation of the initial problem) using the Nash equilibrium and the corresponding mixed optimal strategies. Finally, we transform the solution of the obtained problem to obtain the solution of the initial problem.

We make also a $C++$ program to implement the algorithm presented in the paper.

***Index Terms*— Linear programming, evolutionary game theory, Nash equilibrium.**

I. INTRODUCTION

The classical method to solve the linear programming problem is the SIMPLEX algorithm [10,2,1]. In [1] there is presented a SIMPLEX algorithm when the variables are bordered.

Other methods (iterative ones) to solve the linear programming problem are interior/exterior point methods [2]. These methods have the same advantage as the iterative methods to solve linear systems: they avoid the problem of miss conditioned problem [8].

The ellipsoidal algorithm of Hacian is an exterior point method and solves the linear programming problem [2]

$$\begin{cases} \min c^T x \\ A \cdot x = b \\ x \geq 0 \end{cases} \quad (1)$$

and it's dual

$$\begin{cases} \max b^T u \\ A^T \cdot u \leq c \\ x \in \mathbb{R} \end{cases}. \quad (1')$$

It results that the solutions of the primal and dual problem x^* and u^* must be solutions of the system

$$\begin{cases} A \cdot x = b \\ x \geq 0 \\ A^T \cdot u \leq c \\ c^T x - b^T u \leq 0 \end{cases}. \quad (2)$$

This system can be written [2]

$$S \cdot y \leq \beta, \text{ where} \quad (2')$$

$$S = \begin{pmatrix} A & 0_{m\times m} \\ -A & 0_{m\times m} \\ -I_n & 0_{n\times m} \\ 0_{n\times n} & A^T \\ c^T & -b^T \end{pmatrix} \text{ and} \quad (2'')$$

$$\beta = \begin{pmatrix} b \\ -b \\ 0_n \\ c \\ 0 \end{pmatrix}. \quad (2''')$$

Of course, $y = \left(x^T, u^T\right)^T$ [2]. If we denote by L the size of the problem and if we put $\varepsilon = 2^{-2L}$, we take $\beta_i' = \beta_i + \varepsilon$. The ellipsoidal algorithm is as follows.

Algorithm ellipsoidal

$K \leftarrow 16 * n * (n+1) * L$

$t \leftarrow 0$

$B \leftarrow n^2 * 2^{2L} * I_n$

for $k = \overline{1, K}$

if $S * t < \beta'$ *then*

write 'solution=', t

stop

else

Choose i such that $S_i * t \geq \beta_i'$

$a \leftarrow S_i^T$

$t \leftarrow t - B*a / \left((n+1) * \sqrt{a^T * B * a} \right)$

$B \leftarrow n^2 / \left(n^2 - 1 \right) *$

$\left(B - 2/(n+1) * (B*a) * (B*a)^T / \left(a^T * B * a \right) \right)$

endif

endif

next k

if $k = K$ *then*

write 'no solution'

end

The ellipse $E_k = \left\{ x \in \mathbb{R}^n \middle| (x - t_k)^T \cdot B_k^{-1} \cdot (x - t_k) \leq 1 \right\}$, where t_k and B_k are the vector t and the matrix B at iteration k in the above algorithm, has the volume $Vol(E_k) < 2^{-(n-2)L}$ [2].

An interior point method is the projective algorithm of Karmarkar [2]. Firstly we must reduce the problems (2) and (2') to the standard Karmarkar form [2]

$$\begin{cases} \min c_1^T \cdot x_1 \\ A_1 \cdot x_1 = 0 \\ e^T \cdot x_1 = 1 \\ x \geq 0 \end{cases}, \tag{3}$$

where e has all the components equal to 1.

Next we take $\overline{A} = \begin{pmatrix} A & 0 & 0 & 0 \\ 0 & A^T & -A^T & I \\ c^T & -b^T & b^T & 0 \end{pmatrix}$, $\overline{b} = \begin{pmatrix} b \\ c \\ 0 \end{pmatrix}$ and $y = \begin{pmatrix} x \\ u^+ \\ u^- \\ s \end{pmatrix}$, and we obtain the constraints' system [2]

$$\begin{cases} \overline{A} \cdot y = \overline{b} \\ y \geq 0 \end{cases}. \tag{4}$$

Next we take $L = (m+1)n + \lfloor \log_2 |P| \rfloor + n \cdot \lfloor \log_2 n \rfloor + n + 1$, where $\lfloor \alpha \rfloor$ is the integer part of α, m is the number of constraints of the primal problem and n is the dimension of the variable. We take also $\mu = n \cdot 2^L$, $\overline{A}' = \left(\overline{A} \; 0 \right)$ and $x' = \frac{1}{\mu} \cdot y$. It results that

$$\begin{cases} \mu \overline{A}' \cdot x' = \overline{b} \\ e^T \cdot x' = 1 \\ x' \geq 0 \end{cases}. \tag{5}$$

Because μ has a high value we can multiply the last equation $e^T \cdot x' = 1$ by constants and subtract from the others, and we obtain the restrictions from the standard Karmarkar form with $A_1 = A'$ and $x_1 = x'$. If we add to the matrix A' a null column, to the vector e another component equal to 1 and another component to x' we obtain the matrix $\hat{A}$, the vector e having also all the components equal to 1 and the variable vector $\hat{x}$ with the first components from x'. The initial primal problem has a finite optimum solution if and only if [2]

$$\begin{cases} \min \hat{x}_r \\ \hat{A} \cdot \hat{x} = 0 \\ e^T \cdot \hat{x} = 1 \\ \hat{x} \geq 0 \end{cases} \tag{6}$$

has the minimum equal to 0 (r is the dimension of $\hat{x}$). In this case the first n components of $\hat{x}$ are the components of the solution of the primal problem, next m components are the components of u^+ and next m components are the components of u^-. The solution of the dual problem is $u^+ - u^-$.

The algorithm of Karmarkar solves a problem in the standard form of Karmarkar. Denoting by Π the set of the feasible solution of such linear programming problem, we need a function Φ having the following properties:

1. If $a \in \Pi$, $a > 0$ and $c^T a > 0$ then $\Phi(a) \in \Pi$ and $\Phi(a) > 0$.
2. If the minimum of the problem in the standard Karmarkar form is zero then either $c^T \cdot \Phi(a) = 0$, either $f(\Phi(a)) \leq f(a) - \delta$.

For this we take $f(x) = \frac{(c^T x)^n}{\prod_{i=1}^{n} x_i}$, $\alpha \in \left(0, \frac{1}{2}\right)$ and $\delta = \alpha - \frac{\alpha^2}{1-\alpha}$. Of course, the dimension of the problem's solution is considered to be n. We take also $D = diag(a)$ the diagonal matrix using the given vector a, $c' = D \cdot c$, $B = \begin{pmatrix} A \cdot D \\ e^T \end{pmatrix}$, $c_p = c'^T \left(B \cdot B^T \right)^{-1} D c'$ and $\hat{c}_p = \frac{c_p}{\|c_p\|}$. If we denote by $b' = a - \frac{\alpha}{n} \cdot \hat{c}_p$ and by $b = \frac{D \cdot b'}{e^T \cdot D \cdot b'}$ we define $\Phi(a) = b$.

Algorithm Karmarkar

$a \leftarrow \frac{1}{n} * e$

$K \leftarrow \lfloor 2 * n * L / \delta \rfloor + 1$

if $c^T * a = 0$ *then*

write 'solution is ', a

Stop

else

for $k = \overline{1,K}$

$b \leftarrow a$

$a \leftarrow \Phi(a)$

if $c^T * a = 0$ *then*

write 'solution is ', a

Stop
endif
if $f(a) > f(b) - \delta$ *then*
write 'solution is greater then 0'
Stop
endif
endif

Find $v \in \Pi$ an extremal point such that the cost is less than or equal to $c^T * a$

write 'solution is ', v
end

Genetic algorithms can be also used to solve the linear programming problem. If we have to maximize or minimize a function $f : R \to R$ we consider the population $P \subset R^m$, and the components of $x \in P$ are called chromosomes.

The general structure of a genetic algorithm is the following [7,6]:

1. Generate the initial population.
2. Generate the new individuals of P.
3. Apply genetic mutations on the new individuals.
4. Select a part of the new population.
5. Repeat the steps $2-4$ until it is fulfilled a stopping condition, or for a given number of steps.

The initial population is usually random [7,6,5]. The new generation is generated by the cross-over at the chromosome k: if we have two individuals with the chromosomal chains

$$\begin{cases} x = (x_1,...,x_k,x_{k+1},...,x_m)^T \\ y = (y_1,...,y_k,y_{k+1},...,y_m)^T \end{cases} \quad (7)$$

we obtain the offsprings

$$\begin{cases} z_1 = (x_1,...,x_k,y_{k+1},...,y_m)^T \\ z_2 = (y_1,...,y_k,x_{k+1},...,x_m)^T \end{cases}. \quad (7')$$

The genetic mutations are modifications small enough for a chromosome x_k. These mutations can be chaotic (random) or deterministic, made to improve the objective function (we increase or decrease the value with a small step) [7,6,5]. If the problem is not an optimization one we build a function to maximize it (fitness) or one to minimize it (error).

If we have to minimize a function we order the population increasing on the objective function, respectively we order the population decreasing. Next we remove the last individuals [7,6,5].

The stopping condition is

$$\left| f\left(x^{(n-1)}\right) - f\left(x^{(n)}\right) \right| < \varepsilon, \quad (8)$$

where ε is a given error and $x^{(n)}$ is the best individual in the population at the step n.

The VNS (Variable Neighborhood Search) method is presented in [4]. In fact it is a more general method to solve the problem

$$\begin{cases} \min f(x) \\ x \in S \\ x \in X \end{cases}, \quad (9)$$

where $X \subseteq S$.

The first step in VNS is the VND (Variable Neighborhood Descent), which finds a local optimum:

Algorithm VND
for $l = \overline{1, l_{max}}$
Choose the neighborhood structure $\mathbf{N}_l$
Choose x as initial solution.
repeat
$l \leftarrow 1$
repeat
Find the best neighbor $x' \in \mathbf{N}_l(x)$
if $f(x') < f(x)$ *then*
$x \leftarrow x'$
$l \leftarrow 1$
else
$l \leftarrow l + 1$
until $l = l_{max}$
until no improvement of x
end

Using this algorithm, we use the VNS algorithm [4]:

Algorithm VNS
for $k = \overline{1, k_{max}}$
Choose the neighborhood structure $\mathbf{N}_k$
Choose x as initial solution.
Choose a stopping condition.
repeat
$k \leftarrow 1$
repeat
Generate a random $x' \in \mathbf{N}_k(x)$.
Using the above VND algorithm find a local optimum x'' starting with the initial solution x'
if $f(x'') < f(x)$ *then*
$x \leftarrow x''$
$k \leftarrow 1$
else
$k \leftarrow k + 1$
until $k = k_{max}$
until stopping condition
end

To avoid being blocked in a valley (in a neighborhood of a local minimum) the last neighborhood structure $\mathbf{N}_k$ for $k = \overline{1, k_{max}}$ must fulfill the relation [4]:

$$X \subseteq \bigcup_{k=1}^{k_{max}} \mathbf{N}_k(x) \; \forall x \in X. \quad (10)$$

A particular case of such neighborhood structure is the nested neighborhood structure [4]:

$$\begin{cases} \mathbf{N}_1(x) \subset \mathbf{N}_2(x) \subset ... \subset \mathbf{N}_{k_{max}}(x) \\ X \subseteq \mathbf{N}_{k_{max}}(x) \end{cases} \forall x \in X. \quad (10')$$

Therefore the VNS algorithm explores increasingly far neighborhoods of the current local minimum, with a descent method, and re-centers the search when we find a better solution then the incumbent [3,4].

The stopping condition can be the maximum number of iterations, the maximum number of iterations between two improvements of the local minimum, or maximum CPU time allowed [3].

The VNDS (Variable Neighborhood Decomposition Search) is presented in [3]. The difference between this method and VNS is that instead of searching the local minimum x" starting from x' in the whole S we solve at each iteration a sub-problem in some subspace $V_k \subseteq \mathbf{N}_k(x)$ with $x' \in V_k$.

The linear programming problems that we solve are general enough: the classical constraints problem (the maximization problem in the canonical form) and the classical mixture problem (the minimization problem in the canonical form), and for both problems the involving terms are positives. For this we need some results from game theory [9,10,11] that we will present in the following.

Definition 1. *A game is in the normal form if* $\Gamma = (I, D_i, \pi_i)$, *where* I *is the set of the players,* D_i *is the set of pure strategies of the player* i, *and* $\pi_i : \underset{i=1}{\overset{n}{\times}} D_i \to R$ *is a function with* $\pi_i(d)$ *being the utility of the player* i *if the player* j *uses the strategy* d_j *and* $d = (d_1,...,d_n)$.

Definition 2. *The game is finite if* $I = \{1,2,...,n\}$ *is finite. The game is with complete information if* D_i *are finite.*

Definition 3. *A mixed strategy of the player* i *is a probability distribution* x_i *on* S_i.

We denote by Δ_i the set of mixed strategies of the player i and by $\Theta = \underset{i=1}{\overset{n}{\times}} \Delta_i$. We denote also by $x_i(d_i)$ the probability that the player i uses the pure strategy d_i. Because in the case of non-cooperative games the mixed strategies x_j are independent, we obtain the following formula for the average utility of the player i:

$$u_i(x) = u_i(x_1,...,x_n) = \sum_{d_i \in D_i} \prod_{j=1}^{n} x_j(d_j) \cdot \pi_i(d). \quad (11)$$

Definition 4. $x = (x_i, x_{-i}) \in \Theta$ *is a Nash equilibrium if for any* $y_i \in \Delta_i$ *we have* $u_i(x) \geq u_i(y_i, x_{-i})$.

Theorem 1 ([9,10,11]). *For any finite game with complete information the set* Θ^{NE} *of Nash equilibriums is not empty.*

Definition 5. *A* $2-$player *game with complete information is called bimatriceal.*

In this case $D_1 = \{1,...,m\}$, $D_2 = \{1,...,n\}$, $\pi_1(i,j) = A_{ij}$ and $\pi_2(i,j) = B_{ij}$.

Definition 6. *The game is matriceal if* $A^T = -B$.

An important result from game theory that we use in this paper [9,10,11] is that the Nash equilibrium for the matriceal game given by the matrix A with all the elements positives is computed as follows. We solve first the linear programming problems

$$\begin{cases} \min e^T \cdot x \\ A^T \cdot x \geq e \text{ and} \\ x \geq 0 \end{cases} \quad (12)$$

$$\begin{cases} \max e^T \cdot y \\ A \cdot y \leq e \\ y \geq 0 \end{cases}, \quad (12')$$

where e is a vector with all the components equal to 1. If A is a $m \times n$ matrix, e has m components for the first problem and n components for the second.

The problems (12) and (12') are dual each other, hence the optimal objective function value is the same, and we denote it by z. If the optimal solutions are $\bar{x}$ and $\bar{y}$ the Nash equilibrium is (s_1, s_2) and the game value (the payoff of the first player in the case of Nash equilibrium) is $v = \frac{1}{z}$, where $s_1 = v \cdot \bar{x}$ and $s_2 = v \cdot \bar{y}$.

II. THE ALGORITHM

A method to obtain the Nash equilibrium of a non-cooperative game is to solve first the system of differential equations [11]

$$x'_{ik}(t) = \left(u_i\left(e_i^k, x_{-i}\right) - u_i(x)\right) \cdot x_{ik}(t) \quad (13)$$

with the initial conditions

$$x(0) = x^{(0)}, \quad (13')$$

where $x_i(t) \in \Delta_i$ is the mixed strategy of the player i at the moment t and e_i^k is the pure strategy k of the player i.

Theorem 2([11]). *For any* $x^{(0)} \in \text{Int}(\Theta)$ *the solution* x *of the system of differential equations* (13) *with the initial conditions* (13') *has the property that there exists a Nash equilibrium* $x^* \in \Theta^{NE}$ *such that* $\lim_{t \to \infty} x(t) = x^*$.

For a matriceal game the Cauchy problem (13)+(13') becomes

$$\begin{cases} x'_i = \left(A_i - x^T \cdot A\right) \cdot y \cdot x_i \\ y'_j = y_j^T \left(A \cdot y - A^{(j)}\right) y_j \end{cases} \quad (14)$$

with the initial conditions

$$\begin{cases} x(0) = x^{(0)} \\ y(0) = y^{(0)} \end{cases}, \quad (14')$$

where x is the mixed strategy of the first player, y is the mixed strategy of the second player, A_i is the row i of the matrix A and $A^{(j)}$ is the column j of the matrix A.

We present now the method to solve some nonlinear programming problems. Suppose that the linear programming problem is in the form

$$\begin{cases} \max c^T \cdot x \\ A \cdot x \le b \quad \text{or} \\ x \ge 0 \end{cases} \tag{15}$$

$$\begin{cases} \min c^T \cdot x \\ A \cdot x \ge b \quad , \\ x \ge 0 \end{cases} \tag{15'}$$

where all the involved terms are positive. We can notice that both problems are in the canonical form. First problem is the classical constraints problem, because we have positive benefits and positive quantities for the restricted elements. The second is the classical menu problem because we have positive costs and positive quantities of the menu components. We will solve such problems by the numerical method as follows.

Firstly we reduce the problems to the form (12) or (12') and we consider the matriceal game for which we can find the Nash equilibrium by solving the initial problem and its dual. The way to reduce the problem to a matriceal game is the division of all the elements of the constraints matrix A by the product of the corresponding elements from c and b: we take $A_{ij} \leftarrow \frac{A_{ij}}{b_i \cdot c_j}$. We obtain in this way the matrix of the game in the first case or its transpose in the second case. To remain in the same order of the values of the matrix we can multiply A by a constant equal to the product of the averages of the b values and of c values.

We use next the change of the variable $t = -\ln(1-\tau)$ and we have to solve the system of differential equations

$$\begin{cases} x_i'(\tau) = \left(A_i - x^T \cdot A\right) \cdot y \cdot x_i(\tau) \cdot \frac{1}{1-\tau} \\ y_j'(\tau) = x^T \left(A \cdot y - A^{(j)}\right) y_j(\tau) \cdot \frac{1}{1-\tau} \end{cases} \tag{16}$$

with the initial conditions (14'). These initial conditions are the same as for (14) because for $\tau = 0$ we have $t = 0$. In order to use the theorem 2 we take into account that $\lim_{\tau \to 1} -\ln(1-\tau) = \infty$. The initial conditions must be such that $\left(x^{(0)}, y^{(0)}\right)$ is an interior point of Δ. In our C++ program we take $x_i^{(0)} = \frac{1}{m}$ and $y_j^{(0)} = \frac{1}{n}$ (first player has m pure strategies, and the second one has n pure strategies).

Therefore the Nash equilibrium is $\left(x^*, y^*\right) = \left(x(1), y(1)\right)$. We will compute this Nash equilibrium by numerical methods [8] considering the function $f : R^{m+n+1} \to R^{m+n}$

$$f\left(\tau, \begin{pmatrix} x \\ y \end{pmatrix}\right) = \begin{pmatrix} \frac{\left(A_1 - x^T \cdot A\right) y \cdot x_1}{1-\tau} \\ \cdots\cdots\cdots \\ \frac{\left(A_m - x^T \cdot A\right) y \cdot x_m}{1-\tau} \\ \frac{x^T \left(A \cdot y - A^{(1)}\right) y_1}{1-\tau} \\ \cdots\cdots\cdots \\ \frac{x^T \left(A \cdot y - A^{(n)}\right) y_n}{1-\tau} \end{pmatrix}. \tag{17}$$

In [8] there are presented numerical methods to solve Cauchy problems for differential equations. Therefore there are considered only scalar versions of the function f from (17). For a system of differential equation we divide the interval $[0,1]$ in 500 intervals, and we take $h = 0.02$. Denoting by $\tau_k = k \cdot h$, by $x^{(k)} = x(\tau_k)$ and by $y^{(k)} = y(\tau_k)$ we can solve the Cauchy problem using either the Euler method

$$\begin{pmatrix} x^{(k)} \\ y^{(k)} \end{pmatrix} = \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix} + h \cdot f\left(\tau_{k-1}, \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix}\right), \tag{18}$$

for $k = \overline{1,500}$, the modified Euler method

$$\begin{pmatrix} x^{(k)} \\ y^{(k)} \end{pmatrix} = \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix} + \frac{h\left(g^{(1)} + g^{(2)}\right)}{2} \text{ where} \tag{19}$$

$$\begin{cases} g^{(1)} = f\left(\tau_{k-1}, \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix}\right) \\ g^{(2)} = f\left(\tau_{k-1} + h, \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix} + h \cdot g^{(1)}\right) \end{cases} \tag{19'}$$

for $k = \overline{1,500}$, or the Runge-Kutta method

$$\begin{pmatrix} x^{(k)} \\ y^{(k)} \end{pmatrix} = \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix} + \frac{h\left(g^{(1)} + 2 \cdot g^{(2)} + 2 \cdot g^{(3)} + g^{(4)}\right)}{6}, \text{ where} \tag{20}$$

$$\begin{cases} g^{(1)} = f\left(\tau_{k-1}, \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix}\right) \\ g^{(2)} = f\left(\tau_{k-1} + \frac{h}{2}, \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix} + \frac{h}{2} \cdot g^{(1)}\right) \\ g^{(3)} = f\left(\tau_{k-1} + \frac{h}{2}, \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix} + \frac{h}{2} \cdot g^{(2)}\right) \\ g^{(4)} = f\left(\tau_{k-1} + h, \begin{pmatrix} x^{(k-1)} \\ y^{(k-1)} \end{pmatrix} + h \cdot g^{(3)}\right) \end{cases} \tag{20'}$$

for $k = \overline{1,500}$.

If the matriceal game has the obtained value *v* (the first player's payoff in the case of Nash equilibrium) and the obtained Nash equilibrium using the above methods is (s_1, s_2) the objective function value is $\frac{1}{v}$, the primal solution in the case of maximization is $x_i = \frac{(s_1)_i}{v \cdot c_i}$, and

the dual solution is $y_i = \frac{(s_2)_i}{v \cdot b_i}$. In the case of minimization we switch s_1 and s_2. The above value of v is computed if we do not multiply the elements A_{ij} by the constant equal to the product of the averages of the b values and of c values. In the contrary case the first value of v must be divided first by this constant (the case of our $C++$ program).

III. APPLICATIONS

Example 1. *At a concrete station there are used cement, gravel sand and water to product three types of concrete:* B_1, B_2 *and* B_3.

An unit of B_1 *uses* 8 *units of cement,* 5 *units of gravel, one unit of sand and* 4 *units of water, and it ensures a benefit of* 3 *monetary units.*

An unit of B_2 *uses* 3 *units of cement, one unit of gravel,* 6 *units of sand and* 3 *units of water, and it ensures a benefit of* 3 *monetary units.*

An unit of B_3 *uses* 5 *units of cement,* 4 *units of gravel,* 2 *units of sand and* 3 *units of water, and it ensures a benefit of* 8 *monetary units.*

If the concrete station has 17 *units of cement,* 22 *units of gravel,* 32 *units of sand and* 25 *units of water, compute the maximum benefit and the optimal solution.*

We have to solve the linear programming problem

$$\begin{cases} \max 3 \cdot x_1 + 3 \cdot x_2 + 8 \cdot x_3 \\ 8 \cdot x_1 + 3 \cdot x_2 + 5 \cdot x_3 \le 17 \\ 5 \cdot x_1 + x_2 + 4 \cdot x_3 \le 22 \\ x_1 + 6 \cdot x_2 + 2 \cdot x_3 \le 32 \\ 4 \cdot x_1 + 3 \cdot x_2 + 3 \cdot x_3 \le 25 \\ x_i \ge 0 \end{cases}. \quad (21)$$

By the SIMPLEX method we obtain $\max = 27.2$, the solution $x = (0,0,3.4)^T$, the dual prices $y = (1.6,0,0,0)^T$, the slack variables $(0,8.4,25.2,14.8)^T$ and the reduced costs $(9.8,1.8,0)^T$.

For numerical methods we consider $h = \frac{1}{500} = 0.02$ and the computations are done with five decimals.

If we use the Euler method we obtain $\max = 27.19994$, the solution $x = (0,0,3.39999)^T$, the dual prices $y = (1.6,-0.00001,0,0)^T$, the slack variables $(0.00003,8.40002,25.20001,14.80002)^T$ and the reduced costs $(9.8,1.8,0)^T$.

By the modified Euler method we obtain $\max = 27.19967$, the solution $x = (0,-0.00002,3.39999)^T$, the dual prices $y = (1.6,-0.00002,0,0)^T$, the slack variables $(0.00009,8.40005,25.20012,14.80008)^T$ and the reduced costs $(9.79994,1.79999,-0.00005)^T$.

If we use the Runge-Kutta method we obtain $\max = 27.19739$, the solution $x = (0,0,3.39999)^T$, the dual prices $y = (1.59986,-0.00001,0,0)^T$, the slack variables $(0.00006,8.40004,25.20005,14.80004)^T$ and the reduced costs $(9.79883,1.79957,-0.00074)^T$.

Example 2. *We must create a mixture such that it contains the elements* E_1 (*at leas* 17 *units*), E_2 (*at least* 25 *units*), E_3 (*at least* 28 *units*) *and* E_4 (*at least* 11 *units*). *For this we use the substances* S_1, S_2, S_3 *and* S_4.

An unit of S_1 *contains* 4 *units of* E_1, 2 *units of* E_2, 8 *units of* E_3 *and one unit of* E_4, *and it costs* 5 *monetary units.*

An unit of S_2 *contains* 3 *units of* E_1, 3 *units of* E_2, 4 *units of* E_3 *and* 2 *units of* E_4, *and it costs* 4 *monetary units.*

An unit of S_3 *contains* 7 *units of* E_1, *one unit of* E_2, 2 *units of* E_3 *and* 5 *units of* E_4, *and it costs* 7 *monetary units.*

An unit of S_4 *contains* 4 *units of* E_1, 6 *units of* E_2, 3 *units of* E_3 *and* 3 *units of* E_4, *and it costs* 2 *monetary units.*

Compute the minimum cost and the optimal solution.

We have to solve the linear programming problem

$$\begin{cases} \min 5 \cdot x_1 + 4 \cdot x_2 + 7 \cdot x_3 + 2 \cdot x_4 \\ 4 \cdot x_1 + 3 \cdot x_2 + 7 \cdot x_3 + 4 \cdot x_4 \ge 17 \\ 2 \cdot x_1 + 3 \cdot x_2 + x_3 + 6 \cdot x_4 \ge 25 \\ 8 \cdot x_1 + 4 \cdot x_2 + 2 \cdot x_3 + 3 \cdot x_4 \ge 28 \\ x_1 + 2 \cdot x_2 + 5 \cdot x_3 + 3 \cdot x_4 \ge 11 \\ x_i \ge 0 \end{cases}. \quad (22)$$

By the SIMPLEX method we obtain $\min = 17.92859$, the solution $x = (2.21429,0,0,3.42857)^T$, the dual prices $y = (0,0.02381,0.61906,0)^T$, the slack variables $(5.57143,0,0,1.5)^T$ and the reduced costs $(0,1.4524,5.73907,0)^T$.

For numerical methods we use the same $h = 0.02$, and for all the methods (including SIMPLEX) we use the same precision as in example 1 (5 decimals).

If we use the Euler method we obtain $\min = 18.37677$, the solution $x = (0.86584,0,0,7.02379)^T$, the dual prices $y = (0,0,0.65636,0)^T$, the slack variables $(14.55852,18.87443,-0.00192,10.93722)^T$ and the reduced costs $(-0.25087,1.37457,5.68728,0.03093)^T$.

By the modified Euler method we obtain $\min = 18.54166$, the solution $x = (1.01166,-0.00002,0,6.74172)^T$, the dual prices $y = (0,0,0.65476,0)^T$, the slack variables

$(14.01346, 17.47359, 0.31835, 10.23679)^T$ and the reduced costs $(-0.23806, 1.38097, 5.69048, 0.03573)^T$.

If we use the Runge-Kutta method we obtain $\min = 19.27518$, the solution $x = (1.03345, -0.00001, 0, 7.05398)^T$, the dual prices $y = (0, 0, 0.65496, 0)^T$, the slack variables $(15.3497, 19.39076, 1.42952, 11.19538)^T$ and the reduced costs $(-0.23969, 1.38016, 5.69008, 0.03512)^T$.

IV. CONCLUSIONS

Because the expression $1-\tau$ appears at the denominator in (17) for the last step (computation of $x(1)$ and $y(1)$) we can not use the modified Euler method or the Runge-Kutta method. This because we can not compute $g^{(2)}$ in (19') or $g^{(4)}$ in (20'), hence we use the Euler method for the last step. This is also the reason for that increasing the number of steps does not guarantee a better solution: the components of f become huge because of the denominator, and the increasing of the error can be higher then the decreasing given by small h. When we test the program we can take $n = 1000$ or $n = 100$, but for $n = 500$ we obtain solutions closer to those obtained by the SIMPLEX algorithm.

We notice also that some values for the constraints and for the reduced costs (the constraints of the dual) are negatives. This means that the constraint is not fulfilled (for instance if the constraint is $A_i \cdot x \le b_i$ with A_i the line i of the matrix of constraints, and the constraint value is -0.5, the value of the left side is greater then b_i by 0.5). We can say the same thing for the constraints $A_i \cdot x \ge b_i$ and for the reduced costs. If some of the constraints of the problem are too important to be fulfilled we have to multiply x by a positive constant such that all these constraints become fulfilled. Next we compute the new value of the objective function and the other constraints. We do the same thing with y if some reduced costs are important. If we do this with x and y we compute the objective function for the primal and the dual problem, and next the average of the two values.

The initial solution of the linear programming problem in the standard Karmarkar form is $a_i = \frac{1}{n}$ (the center of the simplex). In the same way we choose the initial conditions of the system of differential equations (16) $x_i^{(0)} = \frac{1}{m}$ and $y_j^{(0)} = \frac{1}{n}$.

The use of systems of differential equations to find a Nash equilibrium is analogous to the mutations in genetic algorithms [6,7]. The difference is that the mutations are only to one chromosome, and, even in applications we apply successive mutations after cross-over these mutations have the same size (0.001 for instance [7]) and the sense of the partial derivative of the fitness function on the considered chromosome. The derivative of the component x_i (which can be looked as a chromosome) in our paper has not only the sense, but also the absolute value of the difference between the payoff using the pure strategy i and that using the mixed strategy x, multiplied by the size of x_j and divided by $1-\tau$. The same thing we can say about y. In fact another difference to genetic algorithms is that instead of a population in our paper we have two individuals of two antagonist population. It is a completion to the genetic algorithms, because we deal with what these individuals do during their life. The genetic algorithms use selection [5,6,7] and our algorithm uses accommodation. Genetic algorithms can be also used to solve differential equations [5].

An open problem is how to reduce more general linear programming problem to our case. Another open problem is to extend our method to the quadratic programming and to the convex programming.

REFERENCES

[1] N. Andrei, *Programare matematică avansată. Teorie şi metode computaţionale*. Ed. Tehnică, Bucureşti, 1999.

[2] A. Bătătorescu, *Metode de optimizare liniară*. Ed. Universităţii Bucureşti, 2003.

[3] P. Hansen, N. Mladenovic and D. Perez-Brito, "Variable Neighborhood Decomposition Search", *Journal of Heuristics*, No. 7, pp. 335-350, 2001.

[4] P. Hansen and N. Mladenovic, "Variable Neighborhood Search", in *Search Methodologies. Introductory tutorials in Optimization and Decision Support Techniques*, Editors E.K. Burke and G. Kendall, Springer, 2005, pp. 211-238.

[5] G.D. Mateescu, "On the Application of Genetic Algorithms to Differential Equations", *Romanian Journal of Economic Forecasting*, No. 2, pp. 5-9, 2006.

[6] G.D. Mateescu, "Algoritmi genetici de optimizare", *Working Papers of Macroeconomic Modeling Seminar*, No. 061002, pp. 21-28, 2006.

[7] G.D. Mateescu, C. Sâman and M. Buneci, "Algoritmi genetici", *Working Papers of Macroeconomic Modeling Seminar*, No. 071402, pp. 21-31, 2007.

[8] G. Păltineanu, P. Matei and R. Trandafir, *Analiză numerică*. Ed. Conspress, Bucureşti, 1998.

[9] A. Ştefănescu, *Competitive Models in Game Theory and Economic Analysis*. Ed. Universităţii Bucureşti, 2000.

[10] A. Ştefănescu and C. Zidăroiu, *Cercetări operaţionale*. Ed. Didactică şi Pedagogică, Bucureşti, 1981.

[11] J. Weibull, *Evolutionary game theory*. MIT Press, 1996.